STRUKTUR DATA NON LINEAR

“BINARY SEARCH TREE”

MODUL PERTEMUAN KE - 5 : DELETE 2 ANAK

DISUSUN OLEH :

Nama : Andreas Nathanael Priambodo

NIM : 215314043

TEKNIK INFORMATIKA FAKULTAS SAINS DAN TEKNOLOGI UNIVERSITAS SANATA DHARMA YOGYAKARTA 2023

A. Soal

Diagram Kelas UML

TreeNode	
- data	: int
- leftNode	: TreeNode
- rightNode	: TreeNode
- parent	: TreeNode
+ TreeNode (int, TreeNode)	: Constuctor
+ insert(int)	: void

Tree	
– root	: TreeNode
+ Tree()	: Constructor
+ insertNode(int)	: void
+ delete(int)	: boolean
+ getCurrent(int)	: TreeNode
+ getPredeccessor(TreeNode)	: TreeNode

Catatan : Tambahkan method set, method get dan method kunjungan (preorder, inorder & postorder) seperti modul 3.

1. Buatlah program BST berdasarkan diagram kelas UML diatas.
2. Buatlah algoritma dan implementasikan untuk method delete 2 anak (predeccessor).
3. Buatlah method main dan eksekusilah program BST dengan memasukan beberapa data dan lakukan penghapusan pohon serta catat perubahannya.

B. Algoritma

- Delete
    1. Jika root sama dengan kosong maka lanjut ke perintah selanjutnya jika tidak maka false
    2. Panggil method deleteHelper dengan menggunakan argument root dan data yang akan dihapus
    3. Jika method deleteHelper mengembalikan node maka data tersebut telah dihapus kembalikan true jika tidak false
- deleteHelper
    1. Jika node null kembalikan null

2. Jika data yang akan dihapus kurang dari data pada node saat ini, panggil deleteHelper dengan argumen node kiri dan data yang akan dihapus.
3. Jika data yang akan dihapus lebih besar dari data pada node saat ini, panggil deleteHelper dengan argumen node kanan dan data yang akan dihapus.
4. Jika data yang akan dihapus sama dengan data pada node saat ini, lanjutkan ke langkah 5. Jika tidak, kembalikan node saat ini.
5. Jika node memiliki kedua node anak, cari predecessor dari node saat ini dengan memanggil getPredecessor(node.getLeftNode()).
6. Set nilai data pada node saat ini dengan nilai data predecessor.
7. Hapus predecessor dengan memanggil deleteHelper dengan argumen node kiri dan data predecessor.
8. Jika node hanya memiliki node anak kiri, set node saat ini menjadi node anak kiri.
9. Jika node hanya memiliki node anak kanan, set node saat ini menjadi node anak kanan.
10. Jika node tidak memiliki node anak, set node saat ini menjadi null.
11. Kembalikan node saat ini.

C. Program

- MainTree

```
package binary_search_tree_copy;

import java.util.Scanner;

public class MainTree {

   public static void main(String[] args) {
      Scanner dtSc = new Scanner(System.in);
      Tree Ob = new Tree();
      Ob.Insert(42);
      Ob.Insert(21);
      Ob.Insert(38);
      Ob.Insert(27);
```

```
        Ob.Insert(71);
        Ob.Insert(82);
        Ob.Insert(55);
        Ob.Insert(63);
        Ob.Insert(6);
        Ob.Insert(2);
        Ob.Insert(40);
        Ob.Insert(12);
        System.out.println();
        System.out.println("PreOrder : ");
        Ob.preOrderTransversal();
        System.out.println();
        System.out.println("InOrder : ");
        Ob.inOrderTransversal();
        System.out.println();
        System.out.println("PostOrder : ");
        Ob.postOrderTransversal();
        System.out.println();
        System.out.print("Search Data : ");
        int search = dtSc.nextInt();
        TreeNode dataSearch = Ob.Search(search);
        if (dataSearch == null) {
            System.out.println("Data not Found");
        } else {
            System.out.println("Data " + dataSearch.getData() + " Found");
        }
    }
}
```

- Tree

```
package binary_search_tree_copy2;


public class Tree {
```

```
private TreeNode root;

public Tree() {
    root = null;
}

public Tree(TreeNode root) {
    this.root = root;
}

public TreeNode getRoot() {
    return root;
}

public void setRoot(TreeNode root) {
    this.root = root;
}

public void preOrderTransversal() {
    preOrderHelper(root);
}

public void inOrderTransversal() {
    inOrderHelper(root);
}

public void postOrderTransversal() {
    postOrderHelper(root);
}

public void preOrderHelper(TreeNode node) {
    if (node != null) {
```

```
        System.out.print(node.getData() + " ");
        preOrderHelper(node.getLeftNode());
        preOrderHelper(node.getRightNode());
    }

}

public void inOrderHelper(TreeNode node) {
    if (node != null) {
        inOrderHelper(node.getLeftNode());
        System.out.print(node.getData() + " ");
        inOrderHelper(node.getRightNode());
    }

}

public void postOrderHelper(TreeNode node) {
    if (node != null) {
        postOrderHelper(node.getLeftNode());
        postOrderHelper(node.getRightNode());
        System.out.print(node.getData() + " ");
    }

}

public void Insert(int in) {
    if (root == null) {
        root = new TreeNode(in);
        System.out.println("Root " + root.getData());
    } else {
        root.insert(in);
    }
}
```

```
public TreeNode getCurrent(int current) {
    TreeNode data = root;
    while (data != null) {
        if (current == data.getData()) {
            return data;
        } else if (current < data.getData()) {
            data = data.getLeftNode();
        } else {
            data = data.getRightNode();
        }
    }
    return null;
}

public TreeNode search(int search) {
    return getCurrent(search);
}

public boolean delete(int data) {
    if (root != null) {
        root = deleteHelper(root, data);
        return true;
    }
    return false;
}

private TreeNode deleteHelper(TreeNode node, int data) {
    if (node == null) {
        return null;
    }

    if (data < node.getData()) {
```

```
            node.setLeftNode(deleteHelper(node.getLeftNode(), data));
        } else if (data > node.getData()) {
            node.setRightNode(deleteHelper(node.getRightNode(), data));
        } else {
            // node dengan 2 child nodes
            if (node.getLeftNode() != null && node.getRightNode() != null) {
                TreeNode predecessor = getPredecessor(node.getLeftNode());
                node.setData(predecessor.getData());
                node.setLeftNode(deleteHelper(node.getLeftNode(), predecessor.getData()));
            } else if (node.getLeftNode() != null) {
                node = node.getLeftNode();
            } else if (node.getRightNode() != null) {
                node = node.getRightNode();
            } else {
                node = null;
            }
        }
        return node;
    }



    private TreeNode getPredecessor(TreeNode node) {
        while (node.getRightNode() != null) {
            node = node.getRightNode();
        }
        return node;
    }
}
```

- TreeNode

```
package binary_search_tree_copy;


public class TreeNode {
```

```
private int data;
private TreeNode leftNode;
private TreeNode rightNode;

public TreeNode(int data) {
   this.data = data;
   leftNode = rightNode = null;
}

public int getData() {
   return data;
}

public void setData(int data) {
   this.data = data;
}

public TreeNode getLeftNode() {
   return leftNode;
}

public void setLeftNode(TreeNode leftNode) {
   this.leftNode = leftNode;
}

public TreeNode getRightNode() {
   return rightNode;
}

public void setRightNode(TreeNode rightNode) {
   this.rightNode = rightNode;
}
```

```
}
```

D. Output

```
run:
Root 10
5 Left 10
15 Right 10
13 Left 15
7 Right 5
20 Right 15

PreOrder :
10 5 7 15 13 20
Berhasil Delete

PreOrder :
10 5 7 15 20

PreOrder :
10 5 7 15 20

PreOrder :
10 5 7 15 20
BUILD SUCCESSFUL (total time: 0 seconds)
```

E. Analisa

- MainTree

  Pada class MainTree ini terdapat Scanner yang digunakan untuk menginput kan data ke dalam console yang sebelumya sudah diimportkan library dari scanner didalam java.util lalu setelahnya ada penginisialisasian object baru untuk class Tree yang Bernama Ob lalu dibawah nya ada pemanggilan method yang ada didalam Ob untuk insert yang digunakan utnuk memasukan data kedalam Tree nya melalui Node lalu dibawahnya pada baris ke 23 terdapat print ke terminal “Search Data : ” lalu inputan yang dimasukan kedalam variable search lalu dibawahnya terdapat variable dari dataSearch yang diisikan dengan Ob.search yang diisikan dengan search variable yang telah diinputkan sebelumnya yang digunakan untuk mencari data yang ada didalam object dari Ob lalu dataSearch tadi diubah menjadi kebentuk if else untuk menentukan apakah data yang dicari ada atau tidak jika ada maka akan mengeluarkan ke terminal Data not Found sedangkan jika ada akan menampilkan Data lalu menggunakan variable dari dataSearch.getData untuk memanggil datanya Found. Dan dibagian bawah dari

inputan angka untuk tree nya terdapat pemanggilan method untuk preOrderTransversal() lalu ada inOrderTransversal() ada juga postOrderTransversal() dimana masing masing ini digunakan untuk mengunjungi node node pada binary tree.

- Tree

  Didalam class tree ini ada attribute yang Bernama root yang berbentuk private yang dimana root ini bertipekan data class TreeNode lalu dibawahnya terdapat constructor yang digunakan untuk pendeklarasian root ini null dan dibawahnya lagi ada kontruktor lagi yang akan terisi apabila akan menambah TreeNode yang sudah ada di main kelas sehingga nanti dapat menggunakan root ini selanjutnya ada setter dan getter dari root dan juga ada method insert dan juga method search untuk inster sendiri berisikan parameter integer Bernama in dan untuk search Sendiri juga menggunakan integer juga tetapi Bernama cari untuk Search akan melakukan algoritma dari Search dan untuk Inser juga akan melakukan algoritma dari Search. Ada tahmbahan untuk sekarang terdapat pre order, in order, dan post order transversal method yang nanti dia memanggil method lain untuk method helper dari masing masing order. Yang pertama untuk preOrderHelper yang berisikan parameter node yang bertipekan class TreeNode didalam bodynya mengecek apakah node nya kosong dan jika kosong akan langsung dikembalikan ke method atasnya yaitu untuk transversal lalu akan Kembali lagi ke main classnya yaitu MainTree selanjutnya untuk bawahnya lagi ada print untuk mencetak data node nya lalu dibawah ada pemanggilan method lagi dan dia memanggil dirinya sendiri atau istilahnya dengan cara rekursif dia akan berulang kali mencari untuk melihat apakah ada leftNode atau tidak dan djika tidak akan Kembali ke rekursif awalan tadi begitu seterusnya. Sama seperti sebelumnya untuk inOrder dengan postOrder yang berbeda hanya dalam urutan print dari node nya karena print disini bisa dianggap sebagai root nya yang akan langsung dicetak. Ada tambahan yaitu untuk Method delete() memiliki parameter data yang merupakan data yang akan dihapus. Pada awal method, dilakukan pengecekan apakah root dari tree-nya tidak null. Jika tidak null, method akan memanggil method deleteHelper() untuk menghapus node yang mengandung data tersebut. Ada juga tambahan Method deleteHelper() merupakan method rekursif yang memproses setiap node dalam tree untuk mencari node yang mengandung data yang akan dihapus. Jika node ditemukan,

dilakukan pengecekan terhadap node tersebut, apakah memiliki satu anak atau tidak memiliki anak. Jika tidak memiliki anak, node tersebut dihapus dari tree. Jika hanya memiliki satu anak, anak tersebut akan menjadi pengganti node tersebut di tree. Jika memiliki dua anak, maka node tersebut tidak dapat dihapus. Jika data yang dicari lebih kecil dari data pada node saat ini, method akan dipanggil lagi pada anak kiri dari node tersebut. Jika lebih besar, maka method akan dipanggil pada anak kanan. Setelah selesai memproses setiap node, method akan mengembalikan node yang sudah diproses. Method delete() memiliki parameter data yang merupakan data yang akan dihapus. Pada awal method, dilakukan pengecekan apakah root dari tree-nya tidak null. Jika tidak null, method akan memanggil method deleteHelper() untuk menghapus node yang mengandung data tersebut. Method deleteHelper() merupakan method rekursif yang memproses setiap node dalam tree untuk mencari node yang mengandung data yang akan dihapus. Jika node ditemukan, dilakukan pengecekan terhadap node tersebut, apakah memiliki satu anak atau tidak memiliki anak. Jika tidak memiliki anak, node tersebut dihapus dari tree. Jika hanya memiliki satu anak, anak tersebut akan menjadi pengganti node tersebut di tree. Jika memiliki dua anak, maka node tersebut tidak dapat dihapus. Jika data yang dicari lebih kecil dari data pada node saat ini, method akan dipanggil lagi pada anak kiri dari node tersebut. Jika lebih besar, maka method akan dipanggil pada anak kanan. Setelah selesai memproses setiap node, method akan mengembalikan node yang sudah diproses. Method insert pada class Tree digunakan untuk memasukkan sebuah node baru ke dalam binary search tree. Jika tree masih kosong (root node belum ada), maka method akan membuat root node baru dengan nilai yang diinput. Jika tree sudah terisi, maka method akan memanggil method insert pada class TreeNode dengan parameter nilai yang diinput, dimulai dari root node. Method getPredecessor adalah method yang digunakan untuk mencari predecessor dari sebuah node pada Binary Search Tree. Predecessor adalah node terbesar pada subtree kiri dari node yang sedang dicari. Method ini akan mengembalikan node predecessor jika ditemukan, atau null jika tidak ditemukan. Pencarian predecessor dimulai dari node kiri dari node yang sedang dicari, dan terus berlanjut ke child node kanan dari setiap node yang dilewati hingga tidak ada child node kanan lagi. Jika node yang ingin dihapus memiliki dua child nodes, maka method ini akan mencari predecessor

dari node tersebut, yaitu node terbesar pada subtree kiri dari node yang akan dihapus. Kemudian, data pada node yang akan dihapus akan diganti dengan data pada predecessor. Setelah itu, predecessor akan dihapus dari Binary Search Tree. Lalu untuk contohnya disini akan menggunakan data {50, 30, 20, 40, 60, 70, 80} yang pertama akan dilakukan adalah mencari predecessor dari node dengan nilai 50 karena yang ingin dihapus adalah 50 disini sebagai peragaan yang memiliki 2 child node. Predecessor dari node 50 ini adalah node dengan nilai maksimum di subtree kiri, yaitu node dengan nilai 40, lalu salin nilai 40 ke node dengan nilai 50. Selanjutnya hapus node 40 dari subtree kiri node dengan nilai 50

- TreeNode

  Didalam TreeNode ini berisikan 3 attribut untuk data yang bertipekan integer lalu ada leftNode dan rightNode yang bertipekan TreeNode lalu ada 1 buah contructor yang berparameter data unguk mengisikan attribute data dan juga ada penginisialisasian untuk leftNode dan juga rightNode menjadi null selanjutnya ada setter dan getter untuk data dan juga leftNode dan rightNode. Method insert pada class TreeNode digunakan untuk memasukkan sebuah node baru ke dalam binary search tree. Jika nilai yang diinput lebih kecil atau sama dengan nilai pada node saat ini, maka method akan mencoba memasukkan nilai tersebut ke sub-tree kiri dari node saat ini. Jika sub-tree kiri belum terisi, maka method akan membuat node baru pada sub-tree kiri tersebut dengan nilai yang diinput. Jika sub-tree kiri sudah terisi, maka method akan memanggil method insert pada sub-tree kiri tersebut dengan parameter nilai yang diinput. Jika nilai yang diinput lebih besar dari nilai pada node saat ini, maka method akan mencoba memasukkan nilai tersebut ke sub-tree kanan dari node saat ini. Jika sub-tree kanan belum terisi, maka method akan membuat node baru pada sub-tree kanan tersebut dengan nilai yang diinput. Jika sub-tree kanan sudah terisi, maka method akan memanggil method insert pada sub-tree kanan tersebut dengan parameter nilai yang diinput. Pada setiap node baru yang berhasil dibuat, method setParent juga dipanggil untuk menunjukkan node yang menjadi parent dari node tersebut.